## **1. Overview**

The **Voice Translation Web App** is a mobile-first browser application that allows users to:

1. Record speech in one language directly from their mobile/desktop browser.
2. Transcribe the recorded speech into text using OpenAI Whisper (gpt-4o-transcribe) model.
3. Translate the transcribed text into a target language using OpenAI GPT-4 via LangChain.
4. Convert the translated text into speech using OpenAI Text-to-Speech (gpt-4o-mini-tts).
5. Play the translated audio back in the browser.

The system is structured as a **React (TypeScript) frontend** and a **FastAPI backend**, connected via REST APIs.

## 2. System Architecture

[User Browser (React Frontend, TypeScript)]

│

▼

Record Audio (MediaRecorder API) → Blob → FormData

│

▼

POST /stt/speech-to-text ──────► [FastAPI Backend]

│

├─ Whisper (gpt-4o-transcribe) → Transcript

├─ LangChain + GPT-4 → Translation

└─ gpt-4o-mini-tts → Translated Speech (audio/wav)

│

▼

Frontend receives:

- Transcript

- Translation

- Audio URL/Base64

│

▼

HTML <audio> element → Playback

## 3. Frontend (React + TypeScript)

### Framework & Tools

* **React (TypeScript)** for UI and state management
* **MediaRecorder API** for microphone input
* **Fetch API** for communication with backend
* **HTML** <audio> for translated audio playback

### Features

* Mobile-first responsive design
* Start/Stop recording with button or **Enter key shortcut**
* FormData upload of audio (WebM/WAV) to FastAPI endpoint
* Displays transcript and translation in clean card components
* Translate and Play Audio buttons

### Core Flow (from App.tsx)

* **Start Recording** → Uses MediaRecorder to capture speech.
* **Stop Recording** → Audio blob converted to .wav or .webm → sent to FastAPI /stt/speech-to-text.
* **Translate Button** → Calls /translate endpoint with transcript + target language.
* **Play Button** → Calls /tts endpoint with translated text, plays audio response.

## 4. Backend (FastAPI, Python)

### Services

#### Speech-to-Text Service (/stt/speech-to-text)

* Accepts audio file (multipart/form-data, field: audio)
* Converts audio to WAV if necessary (via pydub or FFmpeg)
* Transcribes speech using **OpenAI Whisper (**gpt-4o-transcribe**)**
* Returns transcript string

#### Translation Service (/translate)

* Accepts transcript + source/target language
* Uses **LangChain with GPT-4** for translation
* Returns translated text

#### Text-to-Speech Service (/tts)

* Accepts translated text
* Calls **OpenAI TTS (**gpt-4o-mini-tts**, voice="coral")**
* Returns audio stream (WAV/MP3)
* Frontend plays it using <audio>

### Key Libraries

* **FastAPI** → REST backend
* **Pydub / FFmpeg** → audio format conversion
* **OpenAI API** → Whisper STT, GPT-4 translation, TTS
* **LangChain** → Translation orchestration

## 5. File Structure

voice-translation-app/

│

├─ backend/

│ ├─ main.py # FastAPI entrypoint

│ ├─ routers/

│ │ ├─ speech\_to\_text.py # Whisper STT

│ │ ├─ translate.py # GPT-4 Translation

│ │ └─ text\_to\_speech.py # OpenAI TTS

│ └─ requirements.txt

│

├─ frontend/ (React + TS)

│ ├─ src/

│ │ ├─ App.tsx # Main app UI and flow

│ │ ├─ api.ts # API service layer

│ │ └─ components/ # UI components (cards, buttons)

│ └─ package.json

│

└─ README.md

## 6. Deployment

* **Frontend**: Deployed separately (e.g., Vercel, Netlify, or Render static site hosting)
* **Backend**: Deployed on **Render** (FastAPI + Python 3.11+)
* Both apps communicate via public URLs:
  + https://frontend-url.com (React app)
  + https://backend-url.com (FastAPI endpoints)

## 7. Usage Flow

1. User selects input/output language.
2. Click **Start Recording** → Browser records audio.
3. Click **Stop Recording** → Audio file uploaded to /stt/speech-to-text.
4. Transcript displayed.
5. Click **Translate** → Text sent to /translate, translated text returned.
6. Click **Play Translation** → /tts generates audio → plays in <audio> element.

## 8. Design & UI Notes

* **Mobile-first** layout
* Clean, card-based transcript/translation display
* Rounded buttons, touch-friendly padding
* Enter key shortcut to stop recording
* Modern color palette (blues, purples, neutrals)

## 9. Dependencies

### Frontend

* React + TypeScript
* MediaRecorder API (native)

### Backend

* FastAPI
* Pydub / FFmpeg (for audio conversion)
* OpenAI Python SDK
* LangChain

## 10. Future Enhancements

* Auto-detect source language
* Allow audio download of translation
* Dark mode
* Animated recording indicator